Theory of Computation - CSE 105

Context-free Languages
Sample Problems and Solutions

Designing CFLs

Problem 1 Give a context-free grammar that generates the following language over $\{0,1\}^*$:

$$L = \{w | w \text{ contains more 1s than 0s}\}$$

Idea: this is similar to the language where the number of 0s is equal to the number of 1s, except we must ensure that we generate at least one 1, and we must allow an arbitrary number of 1s to be generated anywhere in the derivation. The following grammar accomplishes this task:

$$\begin{array}{rcl} S & \rightarrow & S_1 1 S_1 \\ S_1 & \rightarrow & 0S_1 1 | 1S_1 0 | S_1 S_1 | 1S_1 | \epsilon \end{array}$$

Proof of correctness: it should be clear that this grammar cannot generate any strings not in L. The production for S guarrantees that any string contains at least one 1, and any time a 0 is generated, at least one additional 1 is generated with it. We must argue that the grammar generates all strings with more 1s than 0s. The productions for S_1 generate all strings containing a number of 1s greater than or equal to the number of 0s (proven below). The production for S asserts that any string z in L can be written $z = x1y$ where $N_1(x) \geq N_0(x)$ and $N_1(y) \geq N_0(y)$. This is true: if z begins with a 1, we can say that $z = \epsilon 1y$. If z begins with a 0, we can use a counter which is incremented by 1 for each 0 encountered and decremented by 1 for each 1 encountered, and at some point in the string this counter must become -1 upon encountering a 1 since z contains more 1s than 0s. Let the part of z prior to this point be x and the part of z after this point be y; clearly, this breakdown of $z = x1y$ satisfies the requirements stated above.

Now, to show that S_1 generates all strings z such that $N_1(z) \geq N_0(z)$, the same "counter" argument will work. If z begins with a 0, it must be of the form $z = 0x1y$ where $N_1(x) = N_0(x)$ and $N_1(y) \geq N_0(y)$. If, on the other hand, z begins with a 1, it must either be the case that $z = 1x0y$ where $N_1(x) = N_0(x)$ and $N_1(y) \geq N_0(y)$, or it is the case that $z = 1x$ where $N_1(x) \geq N_0(x)$. Both of these cases are handled by the S_1 transitions.

Problem 2 Give a context-free grammar generating the language

$$L = \text{ the complement of the language } \{a^n b^n | n \geq 0\}.$$

Idea: we can break this language into the union of several simpler languages: $L = \{a^i b^j | i > j\} \cup \{a^i b^j | i < j\} \cup (a \cup b)^* b (a \cup b)^* a (a \cup b)^*$. That is, all strings of a's followed by b's in which the number of a's and b's differ, unioned with all strings *not* of the form $a^i b^j$.

First, we can achieve the union of the CFGs for the three languages:

$$S \rightarrow S_1 | S_2 | S_3$$

Now, the set of strings $\{a^i b^j | i > j\}$ is generated by a simple CFG:

$$S_1 \rightarrow aS_1 b | aS_1 | a$$

Similarly for $\{a^i b^j | i < j\}$:

$$S_2 \to aS_2b|S_2b|b$$

Finally, $(a \cup b)^* b (a \cup b)^* a (a \cup b)^*$ is easily generated as follows:

$$\begin{aligned} S_3 &\to XbXaX \\ X &\to aX|bX|\epsilon \end{aligned}$$

Problem 3) Give a CFG to generate

$$A = \{a^i b^j c^k | i, j, k \geq 0 \text{ and either } i = j \text{ or } j = k\}.$$

Is the grammar ambiguous? Why or why not?

Idea: this language is simply the union of $A_1 = \{a^i b^j c^k | i, j, k \geq 0, i = j\}$ and $A_2 = \{a^i b^j c^k | i, j, k \geq 0, j = k\}$. We can create simple grammars for the separate languages and union them:

$$S \to S_1|S_2$$

For A_1, we simply ensure that the number of a's equals the number of b's:

$$\begin{aligned} S_1 &\to S_1c|A|\epsilon \\ A &\to aAb|\epsilon \end{aligned}$$

Similarly for ensuring that the number of b's equals the number of c's:

$$\begin{aligned} S_2 &\to aS_2|B|\epsilon \\ B &\to bBc|\epsilon \end{aligned}$$

This grammar is ambiguous. For $x = a^n b^n c^n$, we may use either S_1 or S_2 to generate x.

Problem 4 Give a simple description of the language generated by the following grammar in English, then use that description to give a CFG for the complement of that language.

$$\begin{aligned} S &\to aSb|bY|Ya \\ Y &\to bY|aY|\epsilon \end{aligned}$$

Clearly, Y generates $(a \cup b)^*$. S, then, generates strings like $a^n(a \cup b)^* ab^n$ and $a^n b(a \cup b)^* b^n$. Thus we can get strings like $a^i b^j$ where $i > j$, and we can also get strings like $a^i b^j$ where $i < j$, but cannot get $a^i b^j$ where $i \neq j$. Furthermore, we can generate any string beginning with a b or ending with an a, and every string beginning with a and ending with b that is *not* of the form $a^i b^j$. This, then, is exactly the complement of the language $\{a^n b^n | n \geq 0\}$.

A grammar for the complement of this language (which is, of course, just $\{a^n b^n | n \geq 0\}$) is simply

$$S \to aSb|\epsilon.$$

Chomsky Normal Form (CNF)

Problem 5 Convert the following CFG into Chomsky normal form.

$$\begin{aligned} A &\rightarrow BAB|B|\epsilon \\ B &\rightarrow 00|\epsilon \end{aligned}$$

1. First add a new start symbol S_0 and the rule $S_0 \rightarrow A$:

$$\begin{aligned} S_0 &\rightarrow A \\ A &\rightarrow BAB|B|\epsilon \\ B &\rightarrow 00|\epsilon \end{aligned}$$

2. Next eliminate the ϵ-rule $B \rightarrow \epsilon$, resulting in new rules corresponding to $A \rightarrow BAB$:

$$\begin{aligned} S_0 &\rightarrow A \\ A &\rightarrow BAB|BA|AB|A|B|\epsilon \\ B &\rightarrow 00 \end{aligned}$$

3. Now eliminate the redundant rule $A \rightarrow A$ and the ϵ-rule $A \rightarrow \epsilon$:

$$\begin{aligned} S_0 &\rightarrow A|\epsilon \\ A &\rightarrow BAB|BA|AB|BB|B \\ B &\rightarrow 00 \end{aligned}$$

4. Now remove the unit rule $A \rightarrow B$:

$$\begin{aligned} S_0 &\rightarrow A|\epsilon \\ A &\rightarrow BAB|BA|AB|BB|00 \\ B &\rightarrow 00 \end{aligned}$$

5. Then remove the unit rule $S_0 \rightarrow A$:

$$\begin{aligned} S_0 &\rightarrow BAB|BA|AB|BB|00|\epsilon \\ A &\rightarrow BAB|BA|AB|BB|00 \\ B &\rightarrow 00 \end{aligned}$$

6. Finally convert the 00 and BAB rules:

$$\begin{aligned} S_0 &\rightarrow BA_1|BA|AB|BB|N_0N_0|\epsilon \\ A &\rightarrow BA_1|BA|AB|BB|N_0N_0 \\ A_1 &\rightarrow AB \\ B &\rightarrow N_0N_0 \\ N_0 &\rightarrow 0 \end{aligned}$$

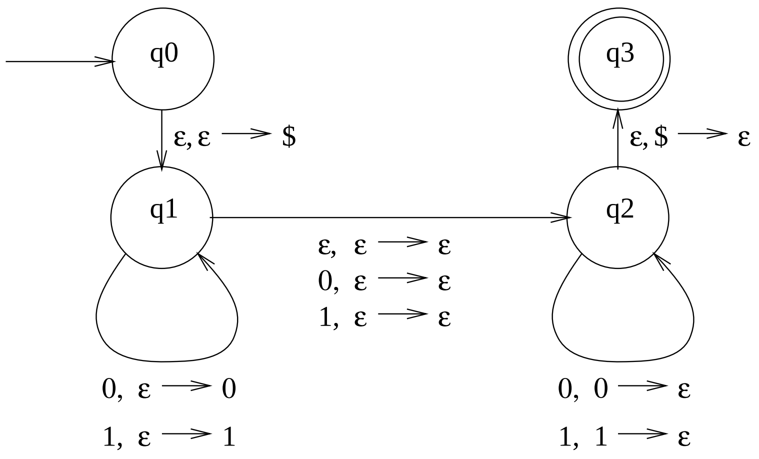
This grammar satisfies all the requirements for Chomsky Normal Form.

Designing PDAs

Problem 6 Give an informal description and state diagram for the language $L = \{w | w = w^{\mathcal{R}},$ that is, w is a palindrom

This is fairly simple: we can push the first half of w, nondeterministically guess where its middle is, and start popping the stack for the second half of w, making sure the second half matches what we pop off the stack. We have to worry about the case where $|w|$ is odd or even, though.

Here is the state diagram:



From the start state q_0, we push a \$ onto the stack to mark its bottom. In state q_1, we push the first half of w onto the stack, not including the middle symbol if $|w|$ is odd. Then we nondeterministically guess where the middle occurs, at which point we can either move to state q_2 without consuming any input if the length of w is even, or simply ignore the middle symbol if $|w|$ is odd. In state q_2, we pop each stack symbol from the stack, ensuring that it matches the current input symbol. Finally, if all goes well, we will reach the end of w with an empty stack (top symbol = \$) and accept. Otherwise the PDA will always crash.

Problem 7 Give an informal English description of a PDA for the langauge $L =$ the complement of the language $\{a^n b^n | n \geq 0\}$.

A PDA for this language can be motivated by the CFG for it. Here is the CFG:

$$\begin{aligned} S &\rightarrow aSb|bY|Ya \\ Y &\rightarrow bY|aY|\epsilon \end{aligned}$$

Recall that this CFG generates strings of the form $a^n b(a \cup b)^* b^n$ pr $a^n (a \cup b)^* ab^n$. All we have to do to accept strings of this form is to push the first n a's onto the stack in state q_1, and nondeterministically switch to a new state q_2 when that is done. At this point we have two branches:

1. If the next symbol is a b, we "flush" that input, go to state q_3, then continue flushing the part of the string corresponding to $(a \cup b)^*$. We nondeterministically guess when this is done and move to state q_4, which pops n b's corresponding to the number of a's that were pushed at the beginning of the string, finally switching to an accept state q_5 if the correct number of b's were matched.

2. If the next symbol was *not* a b, on the other hand, we allow the machine to switch from q_2 to q_6, nondeterministically "flush" the $(a \cup b)^*$ part of the string (in this case our input string must be of the form $a^n (a \cup b)^* ab^n$) then consume the a on the way to state q_4 which as before pops n b's and accepts if everything matches correctly.

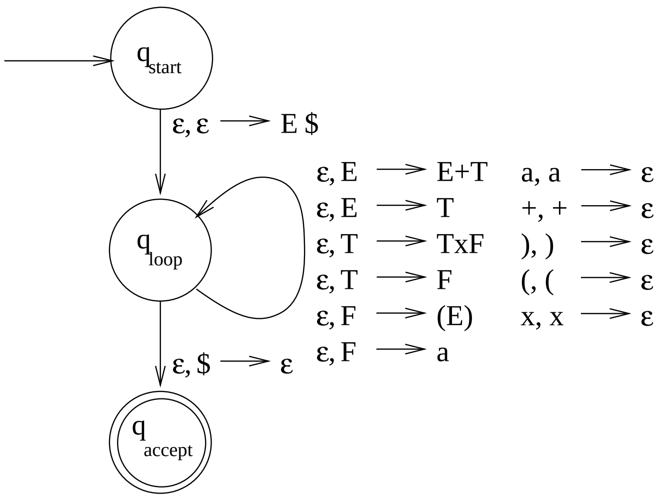
Problem 8 Convert the CFG G_4 given below to an equivalent PDA.

The CFG G_4 is:

$$\begin{aligned} E &\rightarrow E+T|T \\ T &\rightarrow T\times F|F \\ F &\rightarrow (E)|a \end{aligned}$$

Assuming that a shorthand notation allows us to write an entire string to the stack in one PDA step, this task simply reduces to forming transition rules that implement the productions in the grammar.

Here is the PDA:

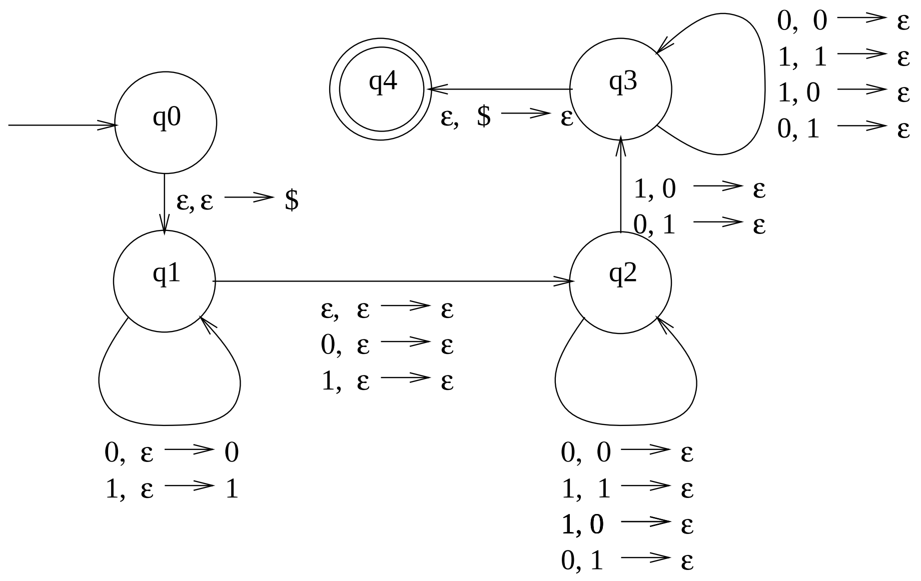


The transitions for the rules of the grammar allow us to nondeterministically replace grammar non-terminals on the stack with their corresponding right-hand-sides; the transitions for the terminals of the grammar $(+, \times,), (, a)$ allow matching of input symbols to grammar terminals. There will be an accepting path through the PDA on string w if and only if w can be generated by the grammar G_4.

Problem 9 Construct a PDA for the language of all non-palindromes over $\{a, b\}$.

We can use the PDA for recognizing palindromes to create a PDA for this language. To change the PDA accepting all palindromes into one that accepts all non-palidromes, we simply insist in the new machine that there is *at least one inconsistency* between the first and second half of input string w. So the new PDA can essentially be the same, except when we are popping symbols off the stack and matching them with inputs, we must make sure that there is at least one a where there should have been a b or vice versa.

Here is the PDA:



We first mark the bottom of the stack with a \$, push the first half of the string (excepting the middle symbol if the string has odd length) onto the stack in q_1, guess nondeterministically where the middle of the string is and switch to state q_2. If $w^{\mathcal{R}} \neq w$, then at some point there will be a mismatch between what is on the stack and in the input; when this is true, the machine can take the transition from q_2 to q_3. Otherwise, any match or mismatch of inputs symbols to symbols on the stack is allowed. Finally, the machine accepts when 1) the input is exhausted and 2) the stack is empty.

Problem 10 Show that $L = \{a^n b^m | m = n^2\}$ is not context-free.

Using the pumping lemma, assume the contrary, and let $s = uvxyz = a^p b^{p^2}$. The lemma says $uv^ixy^iz \in L$ for any $i \geq 0$. Clearly, if either v or y straddles the boundary between a's and b's, pumping s will generate strings not in L. If v and y are composed entirely of a's, then $uv^2xy^2z = a^{p+j}b^{p^2}, 0 < j \leq p$, which is not in L since the number of b's is no longer the square of the number of a's. The same argument holds if $vy = b^k$. The only remaining case is when v is one or more a's and y is one or more b's. If this is the case, then we have $uv^2xy^2z = a^{p+j}b^{p^2+k}$, where $j, k > 0$. But $p^2 + k < (p+1)^2$ since $(p+1)^2 = p^2 + 2p + 1$ and $k < p$ (becaue $k + j \leq p$). Since $p^2 + k$ cannot be *any* perfect square, it certainly cannot be $(p+j)^2$ for any $j > 0$. Since every case results in a contradiction, L is not context-free.

Problem 11 Decide whether $L = \{x \in \{a,b\}^* | N_a(x) < N_b(x) < 2N_a(x)\}$ is a CFL and prove your answer.

L is not context free. We can prove this with the pumping lemma. Let $s = a^{p+1}b^{2p+1}$. Clearly this string is in L. If $vxy = a^j, j > 0$, then $uxz = a^{p+1-k}b^{2p+1}, j \leq k > 0$, which is not in L because the number of b's is more than twice the number of a's. If $vxy = b^j, j > 0$, then $uv^2xy^2z = a^{p+1}b^{2p+1+k}, k > 0$. This string cannot be in L because there are at least twice as many b's as a's. If, on the other hand, vxy contains both a's and b's, then the situation is a little more complicated. If $N_a(vy) > N_b(vy)$ then we can pump down to get uxz, for which $N_a(uxz) = p + 1 - j$ and $N_b(uxz) = 2p + 1 - k$ with $j < k$. But $2(p + 1 - j) = 2p + 1 + 1 - 2j < 2p + 1 - k$, since $1 - 2j < -k$ whenever $j < k$, so uxz is not in L. If $N_a(vx) \leq N_b(vx)$, on the other hand, we can pump up to get a number of b's more than the number of a's: if we use the string uv^pxy^pz, $N_a(uv^pxy^pz) = p + 1 + (p-1)j$ and $N_b(uv^pxy^pz) = 2p + 1 + (p-1)k$ where $k \geq j$. Then $2N_a(uv^pxy^pz) = 2p + 2 + 2j(p-1) < 2p + 1 + (p-1)k$

Problem 12 Write a context-free grammar for the language $L = \{w\#x \mid w^R \text{ is a substring of } x \text{ for } w, x \in \{0,1\}^*\}$.

Solution: Strings in this language share the property that they start with a string w followed by a $\#$, followed by anything, followed by w^R, followed by anything. So we want strings of the form $w\#(0 \cup 1)^* w^R (0 \cup 1)^*$. Let A generate the $w\#(0 \cup 1) * w^R$ part, and let B generate the final $(0 \cup 1)^*$ part. Thus we want derivations that proceed as follows:

$$S \Rightarrow AB \Rightarrow^* wAw^R \Rightarrow^* w\#(0 \cup 1)^* w^R (0 \cup 1)^*.$$

A grammar accomplishing this is:

$$\begin{array}{rcl} S & \rightarrow & AB \\ A & \rightarrow & 0A0 \mid 1A1 \mid \#B \\ B & \rightarrow & 0B \mid 1B \mid \epsilon \end{array}$$

Since the recursion with nonterminal A ends only when the transition $A \rightarrow \#B$, A must generate a string whose beginning and end are mirror images. Since B generates $(0 \cup 1)^*$, the nonterminal A generates all strings of the form $w\#(0 \cup 1)^* w^R$. Note that this also covers the case where $w = \epsilon$. Since A is followed by B in the transition for the top-level nonterminal S, the grammar generates all strings of the form $w\#(0 \cup 1)^* w^R (0 \cup 1)^*$.

Problem 13 Show that $D = \{xy | x, y \in \{0,1\}^*, |x| = |y|, x \neq y\}$ is a context free language.

Solution: any string $z \in D$ must be even length, and its two halves must differ in at least one bit. This means z can be written $z = t0yv1w$ or $z = t1yv0w$ where $|t| = |v|$ and $|y| = |w|$. But this is the same as saying $z = t0vy1w$ or $z = t1vy0w$ where $|t| = |v|$ and $|y| = |w|$. Formulated this way, we can easily write a grammar for the language:

$$\begin{array}{rcl} S & \rightarrow & S_0 S_1 | S_1 S_0 \\ S_0 & \rightarrow & X S_0 X | 0 \\ S_1 & \rightarrow & X S_1 X | 1 \\ X & \rightarrow & 0 | 1 \end{array}$$

Problem 14 Let C be a context-free language and R be a regular language. Prove that the language $C \cap R$ is context-free. Then use the above to show that the language given below is not a CFL.

$$A = \{w \mid w \in \{a, b, c\}^* \text{and contains equal numbers of } a\text{'s, } b\text{'s and } c\text{'s}\}$$

Solution: We have a CFL C and a regular language R and we want to show that $C \cap R$ is context-free. Since C is given to be a CFL we know that there exists a PDA, say M_1, to recognize C. Since R is given to be regular we have a DFA, say M_2, to recognize R. To prove that $C \cap R$ is a CFL we demonstrate a pushdown automaton, call it M, that recognizes $C \cap R$.

The proof is by construction. We construct M from M_1 and M_2. The construction is similar to the proof of showing that the class of regular languages are closed under the union (or intersection) operation on pg. 45 of the text.

Let M_1 recognize C, where $M_1 = (Q_1, \Sigma, \Gamma_1, \delta_1, q_1, F_1)$.

Let M_2 recognize R, where $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$.

Construct M to recognize $C \cap R$, where $M = (Q, \Sigma, \Gamma, \delta, q, F)$.

1. $Q = \{(r_1, r_2) | r_1 \in Q_1$ and $r_2 \in Q_2\}$
2. Σ is assumed the same in M_1 and M_2.
3. $\Gamma = \Gamma_1$
4. δ is defined as follows: for each $(r_1, r_2) \in Q$; each $a \in \Sigma$ and each $b \in \Gamma$ let
$$\delta((r_1, r_2), a, b) = (\delta_1(r_1, a, b), \delta_2(r_2, a))$$
5. $q = (q_1, q_2)$
6. $F = \{(r_1, r_2) | r_1 \in F_1$ and $r_2 \in F_2\}$

Note that the above construction works only because one of the machines being simulated (the DFA M_2 above) does not need a stack. Observe that we may need to maintain 2 stacks if we attempted to simulate 2 PDA's instead, and that a PDA cannot do that.

Now to show that the given language A is not a CFL, we will make the assumption that it is and then derive a contradiction. Under this assumption we are guaranteed (from the part above) that if we intersected some regular language with A, then the resulting language would be a CFL. So if we show that for some regular language R and some language B which is not a CFL that, $A \cap R = B$, then we have derived the contradiction. To see what this R and B might be consider all these languages, A, B and R as capturing "some property". From the definition of A we see that this property is "equality" of a's, b's and c's. For B lets try the canonical example of the language that is not a CFL, viz. $B = \{a^n b^n c^n | n \geq 0\}$. B has the property of "equality" as well as "order" of (zero or more) a's followed by (zero or more) b's followed by (zero or more) c's. Now it is easy to see what we want of R; that R should have the property of "order". This is $R = a^* b^* c^*$ (and we know that this is regular).

Since we have a contradiction, it must be that A is not a CFL.

Problem 15 Let $L_4 = \{w\#x \mid w$ is a substring of $x\}$. Show that L_4 is not a context-free language (CFL).

Solution: In order to show that L_4 is not a CFL, we will proceed by contradiction. Assume L_4 is a CFL. Let p be the pumping length given by the pumping lemma and let $s = a^p b^p \# a^p b^p$. Because s is a member of L_4 and $|s| > p$, the pumping lemma says that s can be split into $uvxyz$ satisfying the following conditions:

1. for each $i \geq 0$, $uv^i xy^i z \in L_4$,
2. $|vy| > 0$, and
3. $|vxy| \leq p$.

For convenience, we also write s as $L\#R$, where L and R stand for the strings to the left and to the right of $\#$, respectively.

First of all, we can note that vy cannot contain $\#$, since uv^2xy^2z would contain more than one $\#$ and would not be in L_4. Then, we can think of three possible cases for the string vy:

- vy contains more symbols from L than from R.
  In this case, $uv^2xy^2z = L'\#R'$ where $|L'| > |R'|$. Therefore, L' cannot be a substring of R' and the entire string is not in L_4.

- vy contains more symbols from R than from L.
  In this case (pumping down), $uv^0xy^0z = L'\#R'$ where $|L'| > |R'|$. Therefore, L' cannot be a substring of R' and the entire string is not in L_4.

- vy contains an equal number of symbols from both L and R.
  In this case, because of conditions 2 and 3 of the pumping lemma, $v = b^j$ and $w = a^j$ for some j such that $1 \leq j \leq p/2$. Therefore $uv^2xy^2z = a^pb^{p+j}\#a^{p+j}b^p$ is not in the language.

Because every possible way of splitting the input into $uvxyz$ yields a contradiction, the initial assumption that L_4 is a CFL is false and the proof is complete.